Prepared (also subject responsible if other)
ETH/RZX Krisztián Pándi

No.
198 17-CNL 113 426 Uen

Approved	Checked	Date	Rev	Reference
ETH/RZXC Elemér Lelik	ETHLEL	2009-04-02	C	GASK2

ROHC Protocol Modules for TTCN-3 Toolset with TITAN, User Guide

Contents

1 Introduction

1.1 Revision history

Date	Rev	Characteristics	Prepared
2006-09-22	PA1	First draft version	ETHESI
2006-11-14	A	Approved after review	ETHESI
2007-01-09	PB1	Updated for TITAN R7	ETHBAAT
2009-03-31	PC1	Updated for TITAN R8	EKRIPND

1.2 About this Document

1.2.1 How to Read this Document

This is the User Guide for the ROHC protocol module. The ROHC protocol module is developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [3] and Function Specification [4].

1.2.2 Presumed Knowledge

To use this protocol module the knowledge of the TTCN-3 language [1] is essential.

To use this protocol module the deep knowledge of the ROHC protocol [5] and [6] is essential.

1.2.3 References

[1] ETSI ES 201 873-1 v.3.1.1 (2005-06)
The Testing and Test Control Notation version 3. Part 1: Core Language

[2] 1/1553-CRL 113 200 Uen
User Documentation for the TITAN TTCN-3 Test Executor

[3] 109 21-CNL113 426-1
ROHC Protocol Modules for TTCN-3 Toolset with TITAN, Product Revision Information

[4] 155 17-CNL113 426
ROHC Protocol Modules for TTCN-3 Toolset with TITAN, Function Specification

[5] RFC 3095:
RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed

[6] RFC 3843:
RObust Header Compression (ROHC): A Compression Profile for IP

1.2.4 Abbreviations

AH	Authentication header
CRC	Cyclic Redundancy Check
ESP	Encapsulating Security Payload
GRE	Generic Routing Encapsulation
IP	Internet Protocol
IR-DYN	IR-DYN ROHC packet
LSB	Least Significant Byte
RFC	Request For Comments
RND	RaNDom bit
ROHC	RObust Header Compression
RTP	Real-time Transport Protocol
TTCN-3	Testing and Test Control Notation version 3
UDP	User Datagram Protocol

1.2.5 Terminology

No specific terminology is used.

1.3 System Requirements

Protocol modules are a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, protocol modules alone do not put specific requirements on the system used. However in order to compile and execute a TTCN-3 test suite using the set of protocol modules the following system requirements must be satisfied:

- TITAN TTCN-3 Test Executor R8A (1.8.pl0) or higher installed. For installation guide see [2]. Please note: This version of the protocol module is not compatible with TITAN releases earlier than R8A. Using the test port with TITAN version earlier than R7E can cause dynamic testcase error due to the different integer number handling.

2 Protocol Modules

2.1 Overview

Protocol modules implement the messages structure of the related protocol in a formalized way, using the standard specification language TTCN-3. This allows defining of test data (templates) in the TTCN-3 language [1] and correctly encoding/decoding messages when executing test suites using the TITAN TTCN-3 test environment.

Protocol modules are using TITAN's RAW encoding attributes [2] and hence are usable with the TITAN test toolset only.

2.2 Installation

The set of protocol modules can be used in developing TTCN-3 test suites using any text editor. However to make the work more efficient a TTCN-3-enabled text editor is recommended (e.g. nedit, xemacs). Since the ROHC protocol is used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant section of [2].

2.3 Configuration

None.

2.4 Useful constants

The protocol module provides some constants that can be used in templates.

ROHC operation modes:
- cg_ROHC_mode_C : mode cancellation
- cg_ROHC_mode_U : U mode
- cg_ROHC_mode_O : O mode
- cg_ROHC_mode_R : R mode

ROHC feedback types:
- cg_ROHC_fbck_type_ACK
- cg_ROHC_fbck_type_NACK
- cg_ROHC_fbck_type_SNACK
- cg_ROHC_fbck_type_reserved

2.5 Usage

2.5.1 Encoding of ROHC messages

The encoding of the ROHC messages depends on the negotiated channel parameters and the current state of the compressor's context. The protocol module does not maintain the context of the compressor by itself, all data must be provided by the user.

The encoding function has a parameter, that holds all the necessary channel parameters and context data needed for encoding (only relevant data is listed):

- LARGE_CID (negotiated parameter)
- the list of the UDP ports, that are handled as RTP flows
- one context descriptor for each active contexts with the following data:
  - ROHC operation mode (U/O/R mode), if applicable
  - LSB of the profile identifier (currently supported: 0, 1, 2, 4)
  - IP chain of the context:
    - version of the IP (4 or 6)
    - RND bit
  - UDP data: a flag whether the UDP checksum is present in the packet

2.5.2 Decoding of ROHC messages

The decoding of the ROHC messages depends on the negotiated channel parameters and the current state of the decompressor's context. The protocol module does not maintain the context of the decompressor by itself, all data must be provided by the user.

The decoding function has a parameter, that holds all the necessary channel parameters and context data needed for decoding (only relevant data is listed):

- LARGE_CID (negotiated parameter)
- the list of the UDP ports, that are handled as RTP flows
- one context descriptor for each active contexts with the following data:
  - ROHC operation mode (U/O/R mode), if applicable
  - LSB of the profile identifier (currently supported: 0, 1, 2, 4)
  - decoded packet type
  - IP chain of the context:
    - version of the IP (4 or 6)
    - RND bit
    - flag indicating whether AH header is present in the current index list
    - flag indicating whether ESP header is present in the current index list
    - flag indicating whether GRE header is present in the current index list
    - length of the AH header data field in case the AH header is received in compressed form
    - flag indicating whether the GRE checksum is present in the GRE header (used only if the GRE header flag is true)
  - UDP data: a flag whether the UDP checksum is present in the packet

The ROHC configuration data is an inout parameter in the decoding function. The decoding function will modify the context data based on the information received in the ROHC packet. The updated configuration can be stored and passed to the decoding function again. This technique gives a basic context management logic, which however has limitations (compared to a real ROHC decompressor):

- there is no translation table management, i.e. list compression can not be used.

2.5.3 Encoding of ROHC feedbacks

The encoding of the ROHC feedback messages depends on the negotiated channel parameters.

The encoding function has a parameter, that holds all the necessary channel parameters needed for encoding (only relevant data is listed):
- LARGE_CID (negotiated parameter)

2.5.4 Decoding of ROHC feedbacks

The decoding of the ROHC feedback messages depends on the negotiated channel parameters.

The decoding function has a parameter, that holds all the necessary channel parameters needed for decoding (only relevant data is listed):
- LARGE_CID (negotiated parameter)

2.5.5 Calculating the CRC values

The f_ROHC_CRC() supporting function can be used to calculate the CRC values used in ROHC:
- CRC-3 or CRC-7 over an IP packet
- CRC-32 over a ROHC segment packet
- CRC-8 over an IR or IR-DYN ROHC packet

The signature of the function is:

f_ROHC_CRC (octetstring, integer) returns integer

The first parameter is the buffer, the CRC is calculated over, the second parameter is the CRC type. The CRC type can be 3, 7, 8 or 32. The CRC is calculated over the whole buffer.

If the CRC type has an invalid value, TTCN_error is thrown.

The function returns the calculated CRC value.

3 Examples

3.1 ROHC packet encoding and decoding

Example ROHC configuration:

```
template ROHC_config t_ROHC_config :=
{
  large_cid := false,
  rtp_ports := { 666, 667, 668 },
  context :=
  {
    {
      mode := cg_ROHC_mode_U,
```

```
      profile := 1,
      pkt := NOPKT,
      ip_ctx := {
        {
          version := 4,
          rnd_bit := false,
          ah_present := false,
          gre_present := false,
          esp_present := false,
          ah_data_len := 0,
          gre_cksum_present := false
        }
      },
      udp_ctx := {
        udp_cksum := true
      }
    }
  }
}

// The configuration is inout
var template v_config := t_ROHC_config;
// Encoded data
var octetstring data;
// The ROHC packet
var v_ROHC_pkt;

// Encoding
data := f_ROHC_enc(v_ROHC_pkt, v_config);

// Decoding. The v_config is modified by the codec.
v_ROHC_pkt := f_ROHC_dec(data, v_config);
```

3.2 ROHC feedback encoding and decoding

The ROHC configuration is the same as in the first example.

```
// Encoded data
var octetstring data;
// the ROHC feedback
var Feedback_data v_ROHC_fbck;

// Encoding
data := f_FBCK_enc(v_ROHC_fbck, v_config);

// Decoding
v_ROHC_fbck := f_FBCK_dec(data, v_config);
```

3.3 ROHC CRC calculation

An example to calculate the CRC-8 over an IR packet.

```
var integer crc;
// Encoded data
var octetstring data;
// The ROHC packet. The CRC field must be zero according to RFC 3095.
var v_ROHC_pkt;

// Encoding
data := f_ROHC_enc(v_ROHC_pkt, v_config);

// Calculate the CRC-8
crc := f_ROHC_CRC(data, 8);

// Write the CRC-8 into the data buffer
data[3] := crc;
```